2

AD-A217 576

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 90-0064	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Intelligent Real-Time Problem Solving: Conceptual Analysis of Issues, Ideas and Results		5. TYPE OF REPORT & PERIOD COVERED final technical 8/1/89-10/31/89
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Yoav Shoham and Barbara Hayes-Roth		8. CONTRACT OR GRANT NUMBER(s) F49620-89-C-0103DEF
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS FQ8671-8901799 5581/A7
11. CONTROLLING OFFICE NAME AND ADDRESS Dr. Abraham Waksman Air Force Office of Scientific Research Bldg. 410 Bolling AFB DC 20332-6448		12. REPORT DATE December 1989
		13. NUMBER OF PAGES 43
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (same)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release;
distribution unlimited.

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Our final report consists of four parts. The first two address the issues as defined by the Air Force in the call for proposals, and as discussed in the kickoff meeting. In the first document, authored by Yoav Shoham and Barbara Hayes-Roth, we propose a neutral framework in which to define the terms and issues involved in IRTPS, and make specific recommendations regarding subsequent stages of the project. The second document, authored by Stanley Rosenschein, Barbara Hayes-Roth and Lee Erman, proposes a similar neutral framework.

We then describe our approach to controlling processes intelligently, centered around the notion of agents. In a document authored by Yoav Shoham a general approach to so-called agent-oriented programming is outlines, and in the fourth document, authored by Barbara Hayes-Roth, intelligent real-time agents are discussed in more detail.

IRTPS Final Report

by

Yoav Shoham and Barbara Hayes-Roth

Computer Science Department

Stanford University

Stanford, CA 94305

Our final report consists of four parts. The first two address the issues as defined by the Air Force in the call for proposals, and as discussed in the kickoff meeting. In the first document, authored by Yoav Shoham and Barbara Hayes-Roth, we propose a neutral framework in which to define the terms and issues involved in IRTPS, and make specific recommendations regarding subsequent stages of the project. The second document, authored by Stanley Rosenschein, Barbara Hayes-Roth and Lee Erman, proposes a similar neutral framework.

We then describe our approach to controlling processes intelligently, centered around the notion of agents. In a document authored by Yoav Shoham a general approach to so-called agent-oriented programming is outlines, and in the fourth document, authored by Barbara Hayes-Roth, intelligent real-time agents are discussed in more detail.

Report on Issues, Testbed, and Methodology for the IRTPS Research Program

Yoav Shoham
and
Barbara Hayes-Roth

Stanford University

October, 1989

1. Overview

AFOSR has posed three questions regarding intelligent real-time problem solving (IRTPS):

(a) What are the important terms and issues in IRTPS?

(b) What characteristics should appear in a community testbed application?

(c) What experimental methodology should guide research in this area?

It is easier to contrast an IRTPS system with other frameworks than to define it. IRTPS systems contrast on the one hand with traditional real-time systems, and on the other hand with classical AI planning systems. With the former they share properties such as responsiveness and timeliness. With the latter they share concepts such as process representation, prediction, and reasoning about resource limitations.

To actually define IRTPS systems we need to give precise meaning to the notions mentioned above, and to others such as reactivity, robustness, flexibility and uncertainty. In section 2 we propose a neutral setting in which these terms can be defined, which is compatible with the framework proposed in another workshop paper, "Notes on Evaluating Methodologies for IRTPS Systems," by Stan Rosenschein, Barbara Hayes-Roth, and Lee Erman. In section 3 we introduce several intuitive notions regarding IRTPS requirements and define them in terms of the proposed framework. In section 4 we briefly outline our particular perspective on IRTPS systems, that of *real-time intelligent agents* (on which we say more in two attached documents). In section 5 we give recommendations regarding the testbed environment. Our recommendations regarding experimental methodology appear in the above-mentioned paper and are not repeated here. In section 6 we offer a particular recommendation for structuring AFOSR's IRTPS Research Program.

2. A Computational Model for IRTPS Systems

This section discusses how the modeling of a computational device and its environment, and draws on an analogy with control theory.

Franz Reuleaux' book of 1876, The Kinematics of Machinery, is considered to have laid the foundations to modern kinematics. His insight was the following. In order to reason about the possible motions of physical objects in space, both rotation and translation, one should focus not on the single object but on *pairs* of adjacent objects, the *kinematic pair*. The fundamental object of investigation, proposed Reuleaux, was the interaction between physical objects, the *joint*. The basic question, given a joint between two objects in a specific geometric configuration, is what translational and rotational freedom they allow each other. Reuleaux was able to give a general answer to the question (which since then has been refined and extended).

A similar view of computation is possible, which focuses on the interaction of a machine with its environment. Furthermore, as in the case of kinematics, one need not speak of a machine as one sort of object and its environment as another; instead they can be viewed symmetrically as interacting machines, the *informatic-pair*. A theory of any sort of computation now becomes a theory about the interaction between machines.

A few simple definitions are probably in order at this point. We first define an automaton to consist of several inputs and outputs, an internal state, and a transition function. The transition function is the only nontrivial aspect of the definition: it specifies that the state and output of the machine at each time t are determined by its state at time $t-delta_S$, and its input at time $t-delta_I$. In the following we assume DT, a set of nonzero durations. In the context of an IRTPS system, we take DT to be the positive real numbers.

Definition 1. An *automaton* (also called a machine) is a tuple

$$[S,I,O,F, delta_S, delta_I]$$

where S is a set of states, I is a (possibly empty) set of n-ary tuples of input values (for some fixed n), O is a set of m-ary tuples of output values (for some fixed m), and F is a transition function $F: S \times I \dashrightarrow S \times O$, and $delta_S, delta_I$ are respectively the state and input lags.

Thus, if the machine's state at time $t-delta_S$ is s_1 and its input at time $t-delta_I$ is i_1 and furthermore $F(s_1,i_1)=(s_2,o_2)$, then at time t the machine will be in state s_2 and have output o_2.

With the exception of the temporal aspect, this is a standard construct in system-modeling disciplines. It is a very general model. Although we will restrict the discussion in this manuscript to finite and deterministic automata, we allow in general infinite sets of internal states and stochastic transition functions.

In general we cannot speak of the behavior of the automaton, since that depends on the input. The one case in which we can do it is when the set of inputs is empty.

Definition 2. An automaton with an empty set of inputs is called *closed*; otherwise it is *open*.

For a closed automaton we can assume a unary transition function, one that depends only on its internal state. Given an initial history of the closed automaton's internal states and outputs, we can define the *trace* [or *run* or *behavior*] of the automaton at all future times in the obvious way.

The systems we build, whether they are low-level process controllers or high-level planners, are open automata; we can model their behavior only in conjunction with their environment, which, as was mentioned, will be viewed as simply another automaton.

We first define what it means to *wire* together a collection of automata. Intuitively, when we wire such a collection we connect some inputs to some outputs: an input is connected only to an output of another machine, not necessarily all inputs and outputs are connected, and at most one output is connected to each input and vice versa. (It may seem natural to allow connecting more than one input to a single output. We disallow it because it makes for cleaner semantics of composite machines below. However, we allow a machine to have several outputs that always carry the same value.)

Such a wired set of automata induces a new automaton. Intuitively, the new states are the cross-product of the old states, the new inputs are the union of the old ones that have not been connected to an output, and the new outputs are the union of the old outputs that have not been connected to an input. The only nontrivial aspect is the definition of the new transition function. The subtlety is in the timing. Suppose we manage to catch the collection of automata at a moment when all of them just switched to a new state, which by definition is a new state for the composite machine. When is the next state-change of the composite machine? Intuitively, the composite machine will undergo state transitions at several times, corresponding to the different delays of the individual machines. However, by definition any automaton has a single pair of delays associated with it. Indeed, we will define the composite automaton to change state as soon as any of the individual machines do. In other words, the delay time of the composite automaton is the minimum among the delay times of all individual automata. However, in the new state of the composite automaton each individual automaton will switch to a new "intermediate" state, which will record the remaining delay time of that individual machine.

Definition 3. (Formal definition is omitted.)

One specific kind of wiring is of particular importance, and that is the wiring of two machines into an *informatic pair.*

Definition 4. An *informatic pair* is a pair of automata wired together so that every input of one automaton is wired to an output of the other machine, and vice versa.

By definition, therefore, an informatic pair induces a closed automaton, whose trace is well-defined.

3. *Formal Definitions of Intuitive IRTPS Concepts*

We propose the model outlined in the previous section as one in which to couch discussion of IRTPS issues. Specifically, we propose viewing the IRTPS system as a machine wired to the environment. The various considerations that have been raised in the past can be given precise meanings in terms of the class of environment-machines being considered, their internal structure (which the "intelligent" system will be able to exploit), their wiring to and from the IRTPS (assumptions about input and output), and the internal structure of the IRTPS itself.

We do not propose a comprehensive list of criteria for IRTPS systems, only the setting in which to define them. We hope to reach a concensus of sorts on the criteria during the workshop itself. The most general formulation of them presumably is a some maximization of a utility function over time. Such a requirement would have to specify the class of environments being considered, and the utility function. This is sufficiently general that it's probably both correct and useless (though others may wish to correct us). What we will do instead is pick a few keywords that have been used in the past, and attempt to give them a more-or-less precise meaning in our model. In most cases, these definitions fall short of fully capturing the intuitions behind the terms;

we hope to improve these definitions as well as formulate new ones for other important terms at the workshop. Nonetheless, even these partial definitions contribute to our understanding of these terms by showing that some of them are neutral specifications of the performance required of an IRTPS system, while others are specifications of the internal workings of a particular IRTPS system architecture. (See related discussion in the workshop paper "Notes on Evaluating Methodologies ...")

Timeliness

Intuitively we require that an IRTPS system not only produce correct or useful output, but to do so at the right time. This requirement translates in our framework to a restriction on delays allowable between certain outputs of the environment-machine and its reaching certain other states later. In the simple form the restrictions are absolute, and in the more sophisticated form they are stochastic, allowing occasional harmful long delays in exchange for extra speed at most other times. (From here on we'll talk only of absolute requirements, understanding that the definitions can be extended to the statistical case.)

Some of the other notions below are mentioned in service of timeliness.

Internal Clock

Intuitively, this is a requirement that the IRTPS have a notion of the passage of time in the real world. In our framework this translates to a requirement that some component machine of the IRTPS change at a constant rate.

Recency

The IRTPS should not fall behind real time to handle a backlog of inputs and it should not operate on seriously out of date inputs--unless it has explicitly decided to do so. In terms of our framework, the probability of an environmental state change influencing an IRTPS system state change should decrease rapidly to 0 over time. If the IRTPS sytem has internal structure, then the probability that state changes in its components will affect one another should decrease rapidly to 0 over time.

Guaranteed Cycle Time

Intuitively, we require that the IRTPS never get "lost in thought" for an unbounded period. In our framework this translates to upper bounds on the state and input delays (*deltaS* and *deltaI*) of the IRTPS-machine.

Unpredictability

Intuitively, there is a significant number of environment state and output changes that the IRTPS system cannot predict. In terms of the framework, this means the rate of environmental state transitions varies widely.

Asynchrony

Given this unpredictability, an IRTPS system must receive inputs when the environment produces them and not on any arbitrary schedule. To make sense of this in terms of our framework, we need to model the IRTPS system as a wired set of sub-automata, including some for input reception. Then, the rate of state changes in input reception components must be independent of the rate of state changes in other components.

Data Glut

Intuitively, the IRTPS system will be overwhelmed with data, and will be able to act on only a small part of it. In our framework means that the wiring between the IRTPS system and the environment be dense, with many more input changes to the IRTPS machine than state changes at any given period.

Selectivity, Intelligent Data Filtering

Intuitively, in order to cope with this data glut, the IRTPS cleverly chooses which inputs to process. Defining intelligent data filtering in terms of our framework will depend upon the internal structuring of the machine as a set of wired sub-automata and specification of dependencies among their state changes.

Uncertainty, Noise

Intuitively, the data and world model available to the IRTPS system will be partial, approximate, and sometimes plain wrong. In our framework this translates into imperfect correlations between inputs to and states of the IRTPS, on the one hand, and states of the environment, on the other hand.

Modeling, Prediction, Foresight

Under some architectures, the IRTPS system will have a model of the enviroment, which it can use to predict the future and thus guide its actions. In our framework this is captured by a requirement that some component of the IRTPS system have an internal structure that stands in correlation to the internal structure of the environment. Furthermore, this component must influence state transitions of the IRTPS.

Robustness, Graceful Degradation

Intuitively, as enviroments get more demanding, the performance of the IRTPS system should deteriorate only gradually. To define this concept in our framework, we would have to define measurements on demandingness of the environment and quality of performance of the IRTPS. For example, demandingness of the environment might be measured in terms of number of state variables or rate of state changes. Quality of performance would depend upon application-specific utility functions. Then graceful degradation implies that small changes in demandingness should produce small changes in quality.

4. Perspective on Real-Time Intelligent Agents

We have attempted so far to use as neutral a language as possible, free from terms for which different researchers have conflicting intuitions, or from presuppositions about possible solutions. We now deviate from this restriction and talk briefly about our perspective on real-time intelligent agents.

The term agents is used so much nowadays that it has become meaningless without reference to some particular notion of agenthood. What we mean by this term is a system that is embedded in a temporal framework, and about which one can talk in terms of its having knowledge and beliefs, desires and goals, reasoning capabilities, resource limitations, and similar other mentalistic-sounding terms.

We see several related advatages to dragging this notion of agenthood into the context of IRTPS. Among other things, it allows us to:

* ascribe to the system knowledge of the environment and the related abilities to interpret past events and to predict and plan for future events;

* talk of the goal the system has at any point in time, and the knowledge it needs in order to achieve the goal;

* design and explain the IRTPS as making tradeoffs among different goals, having knowledge of its reasoning resources, and making decisions such as whether to spend time incorporating new data into its world model.

In general, the agent level gives formal meaning to the "I" in IRTPS. If in addition we view the environment as containing other agents we have additional benefits. For example, we can:

* talk of the IRTPS having knowledge of the goals, knowledge and capabilities of these other agents;

* specify the class of environment in which the IRTPS is to function at least partially in terms of what agents it contains, whether they are hostile or friendly, what their capabilities are, etc.

Thus, "agent" is a useful concept to have, both in specifying the problem and (especially) in specifying architectures.

In order to make engineering sense of this concept, however, we must do at least the following:

1. Define the notions we associate with agents, such as knowledge and goals, including their temporal dimension. For example, what does it mean for a process controller to "know" that it "needs" to lower the pressure in the chamber within five seconds? What does it mean for the agent to "believe" that doing so will prevent an explosion?

2. Explain the connection between these high-level notions and low-level behaviors like sensing a signal or actuating an effector.

3. Explain the special constraints placed on agents by real-time environments.

We have both made some progress in this direction, which we describe briefly in two attachments to this paper. "Agent-Oriented Programming" outlines a general approach to treating information systems as formal agents, and some related work that is taking place in Stanford's Robotics Lab. "Research on Adaptive Intelligent Systems" outlines architectural work and related applications being developed in Stanford's Knowledge System's Laboratory.

5. Recommendations regarding Testbed

5.1 Desirable Features of Testbed Environments

Testbeds should include simulated IRTPS environments both for development purposes and for controlled experiments. For verification, it will be necessary to test at least some scenarios in real environments, for example involving interface to a real vision module or to a real process controller.

Testbed simulations should provide appropriate sensors and effectors for use by application systems. They should be factorable--provide "black box" solutions to component tasks--so that researchers can choose to address only those parts of the application problem that interest them. Simulations should be tunable--permit constant factor speed modulations--so that researchers can ignore conventional issues of efficiency and concentrate on more fundamental research issues. Finally, simulations should be instrumented--equipped with meters on important performance variables--so that researchers can analyze the different aspects of performance of their application systems.

Testbeds also should provide a suite of modular application scenarios. Each scenario should be issue-oriented, stressing a particular aspect of intelligent real-time problem solving, such as hard real-time constraints, conflicting sensor data, need for synchronization, etc. Each scenario should include a simulation controller to "play" the scenario and an associated knowledge base (in some neutral representation scheme) for use by application systems in handling the scenario. There should be several variations on each scenario. Base scenarios are for use by researchers in developing aspects of their application system to address the scenario issue. Standard test scenarios should exhibit the same issue-related phenomena as the base scenario, but differ from it on incidental features. For example, a standard test scenario for hard real-time constraints might different from the base scenario on which event carries the constraint, when it occurs in the scenario, what other events occur in the scenario, etc. Standard test scenarios will allow researchers to determine whether they have developed problem-independent approaches.

Stress test scenarios also should correspond to the development scenarios, but they should exhibit extreme cases of the issue-related phenomena. For example, a stress test scenario for hard real-time constraints might impose much shorter time constraints or introduce many more competing events. These scenarios will allow researchers to determine how their approaches degrade under extreme conditions. These different types of scenarios merely illustrate the kinds of scenarios we think would benefit research in an IRTPS testbed.

5.2 Desirable Features of Testbed Domains

An IRTPS testbed domain should exhibit the general task characteristics identified in section 3. To take a few examples, we are interested in domains in which:

* it is not feasible to exhaustively sense interesting features of the environment;

* situation assessment requires interpretion of sensed data and fusion of data from multiple sensors;

* it is not feasible to enumerate every condition the IRTPS will encounter;

* the environment is orderly enough to permit probabilistic prediction of future events;

* coordinated courses of action are sometimes superior to sequences of locally determined actions;

* events vary in the deadlines associated with effective responses;

* multiple goals vary in importance;

* a broad range of relevant knowledge is available;

* explanations of phenomena and rationales for behavior are required.

Because different application domains may differentially emphasize particular subsets of these features, it would be preferable to identify at least two or three testbed applications to avoid artificially skewing research activities toward an arbitrary subset of relevant issues. Moreover, we do not believe that an IRTPS testbed application should replace other applications being studied in the research community, but rather that a diversity of applications promotes a more complete exploration of the space of relevant research issues and solutions.

6. Recommendations for Structuring the IRTPS Research Program

Three programmatic objectives motivate our remarks on structuring the IRTPS Research Program:

* to stimulate new IRTPS research by providing environments in which new investigators could begin to study IRTPS issues without the overhead entailed in developing one's own application;

* to facilitate interactions and exchange of results in the IRTPS community by providing a communications medium in which investigators could demonstrate approaches originally developed in diverse application domains that are unfamiliar to their colleagues;

* to permit a more scientific approach to IRTPS research by providing a controlled environment for comparative evaluation of competing approaches.

As discussed at the first AFOSR workshop on IRTPS, development of a high-quality and easily accessible testbed application that meets all of the requirements would be an expensive and time-consuming task. If we aim for a diversity of testbed applications, as recommended in this paper, the cost rises proportionately. At the same time, this effort would be redundant with efforts already underway by individual researchers to develop a variety of interesting IRTPS testbed applications for their own work, for example:

* Hayes-Roth's simulation of the intensive care environment;

* Hayes-Roth's simulation of GaAs crystal growth by MBE;

* Cohen's simulation of fire-fighting in Yellowstone Park;

* D'Ambrosio's simulation of wilderness exploration;

* Lesser's simulation of vehicle fleet maneuvers;

* SRI's Flakey the robot;

* Latombe's Gofer robots.

Given AFOSR's budget and timetable for this research program, developing a new community testbed application does not seem to be the best use of scarce resources at this time.

As an alternative, we recommend that AFOSR use its IRTPS Research Program to encourage empirical research on testbed applications currently being developed in the community, with particular interest given to two kinds of proposals:

(a) Proposals that offer, in addition to new empirical research, to deliver a testbed version of their application domain suitable for use by other members of the community;

(b) Proposals that offer to comparatively evaluate alternative architectures through controlled experiments within their application domain.

This structuring of the program would address all three progammatic objectives:

* Testbed applications obtained under the first kind of proposal would stimulate new IRTPS research by providing environments in which new investigators could begin to study IRTPS issues.

* These testbed applications would facilitate interactions and exchange of results in the IRTPS community by providing a communications medium in which investigators could demonstrate different approaches.

* Research conducted under the second kind of proposal would specifically include comparative evaluation of competing approaches in a controlled environment.

This structuring of the program would offer two additional advantages:

* It would offer individual researchers more latitude in choosing an application domain for their work.

* It would guarantee exploration of a diverse set of application domains and a broad range of IRTPS issues within the research community.

Notes on Methodologies for Evaluating IRTPS Systems*

Stanley J. Rosenschein (Teleos Research)
Barbara Hayes-Roth (Stanford University)
Lee D. Erman (Cimflex Teknowledge Corp.)

30 October 1989

Abstract

We propose a framework for modeling intelligent real-time problem-solving systems embedded in an environment. Within this framework, measurements may be defined on the system and on the environment, and particular measurements may be designated for judging the performance of the system. Although this framework supports analytical evaluation, we concentrate on its use for experimental evaluation, especially for evaluating and comparing system architectures. This framework also provides a basis for formalizing various requirements terms, such as "reactivity" and "graceful degradation".

1 Introduction

Intelligent real-time problem-solving systems (IRTPSs) are embedded computer systems that interact with their environments in a continuous fashion, sensing asynchronous events and acting in ways designed to satisfy certain goals. Instances of such systems include intelligent robots, factory control systems, avionic systems, and medical monitoring systems. Many software

*This work was supported by AFOSR Contract F49620-89-C-0129 and AFOSR Contract F49620-89-C-0117.

boundary should be placed so as to distinguish between a proposed architecture and the environment in which it is claimed to be effective. Then we can evaluate the relationships between properties of the architecture as manifested in S and properties of E. In particular, note that the $S - E$ boundary need not correspond to the boundary between a "complete agent" and its environment, but may correspond to the boundary around any "partial agent" of interest. For example, to evaluate a complete agent architecture, the $S - E$ boundary should encompass all perception, reasoning, and action elements. But to evaluate a perception architecture, the $S - E$ boundary should more tightly encompass only perceptual elements, with any reasoning or action elements treated as part of the environment. We might often choose to treat particular sensors and effectors as elements of E. As discussed below, for a given placement of the $S - E$ boundary, we will be attempting to attribute properties in the environment to the behavior of particular IRTPSs and, by inference, to their underlying architectures.

3 Measurement and Utility

In order to describe the effectiveness of an IRTPS architecture in controlling aspects of the environment, it is necessary to identify measurements that can be made and how those measurements will be interpreted to determine utility.

A measurement is any function of state values. Measurements can be made within a state or over sets of states or runs.

Under the above model of embedded systems, for a given $S - E$ boundary, measurements on E are distinguished from measurements on S. Measurements on E describe the dynamic properties of the environment, some of which are determined by processes internal to the environment and others of which are influenced by the behavior of S in E. The latter sorts of measurements are distinguished and used to assess the effectiveness of S in determining properties of E under various conditions. These assessments may be in absolute terms or relative to alternative Ss. Measurements on S describe the dynamic properties of the IRTPS, some of which are determined by processes internal to it and others of which are determined by the impact of E on S. These measurements are used help to explain the performance of S and its (in)effectiveness in determining properties of E in terms of its underlying ar-

The choice of utility measures may be specific to the $S - E$ boundary placement. They certainly will be specific to the purpose of the evaluation.

To describe system $S_1(u)$ parametrically with respect to various measurements of utility against fixed (parametric) environment $E(v)$ amounts to characterizing the expected utility of $S_1(u)$ as a function of $\langle u, v \rangle$, using a variety of techniques, some of which are described below. Similar methods can be used to compare two similarly parametrized systems, $S_1(u)$ and $S_2(u)$, fixed (parametric) environment $E(v)$.

4 Evaluation

In principle, there are many ways a proposed IRTPS design might be evaluated. The methods fall into two general categories: *analytical* and *experimental*. Because each of these methods has its advantages and drawbacks, a thorough evaluation often requires both. In the first section below, we briefly mention some of the advantages and disadvantages of analytical methods. The following section treats experimental methods, which are the focus of this document, in more detail.

4.1 Analytical Evaluation Methods

One method of characterizing system performance is by establishing certain of its properties through analytical techniques. These techniques draw on relevant mathematical methods and amount to proving theorems.

The main advantage of the analytical approach is the ability to establish with mathematical certainty very general or universal statements about entire classes of behaviors and phenomena far too numerous to enumerate in explicit detail. For instance, it might be shown mathematically that a certain undesired situation can *never* arise given the nature of the environment and the control system, or that when a triggering event occurs a response is *always* generated within a certain time period. Results of this kind can be very powerful and can give us great confidence in the systems we design.

Analytical approaches are not without their drawbacks, however. The primary drawback is that the complexity of the systems being modeled gives rise to intractable mathematical models that often resist analysis. In an attempt to render the models tractable, simplifications are often made which of their performance on a small number of environmental scenarios.

Drawing general conclusions from a small number of observed instances is a risky business. A given S realizes an abstract architecture, A, in a particular implementation, I, and instantiates it for a body of knowledge, K. Architectures themselves are complex artifacts, differing in both theoretically interesting variables (e.g., knowledge representation, inference procedures, control mechanism) and incidental variables (e.g., implementation details, execution environment). Similarly, different environmental scenarios differ in a great many variables (e.g., frequency of important events, distribution of deadlines, amount of interpretation required, predictability of events). As a consequence, any given observation of the performance of a given architecture on a given environmental scenario is likely to reflect the combined effects of many such variables.

Controlled experimentation is an attempt to reduce as much as possible the incidental variability in a set of observations in order to: (a) obtain a reliable account of particular effects of a particular set of theoretically interesting variables; and (b) rigorously bound the class of situations in which those effects can be expected to obtain. In a controlled experiment, one or more "independent variables" are manipulated and their distinctive effects on one or more "dependent" variables are measured. In the present context, we will typically be manipulating independent variables representing a proposed architecture within S and measuring dependent variables representing the performance of interest within E. Other variables, including both S and E variables, are "controlled" to avoid confounding their effects with those of the independent variables. We also will often evaluate the performance of a fixed S as a function of various E scenarios; for this, we keep S fixed and the dependent and independent variables are in E.

Some controlled variables are simply held constant, while others are systematically manipulated or randomly sampled to provide a basis for generalization. In the domain of IRTPS systems, there are a variety of important variables we may wish to control:

S **variables:**

- sensors
- effectors
- knowledge available

might draw certain conclusions with high confidence, for example: (a) for critical events in all conditions, control mechanism A produces a higher rate of correct answers within deadline than control mechanism B (97% vs. 75%); (b) for non-critical events in all conditions, control mechanism B produces a higher rate of correct answers within deadline than control mechanism A (75% vs. 65%); (c) as the frequency of events increases (1→50 events per unit time), control mechanism A's performance on critical events degrades slowly (100→95%), while its performance on non-critical events declines dramatically 90→40%); and (d) as the frequency of events increases, control mechanism B's performance declines significantly for both critical and non-critical events (95→50% in both cases). Given a particular set of utility functions–in particular, valuing critical events more highly than non-critical events–we might conclude from these results that control mechanism A is "better" than control mechanism B because it provides "better" performance overall and a "better" degradation profile.

Control of variables determines what kinds of conclusions an experiment can support. For example, observing the performance of S_1 and S_2 on a single scenario, O_1, in a single environment, E_1, permits only conclusions about the comparative effectiveness of S_1 and S_2 on scenario O_1. Observing performance on a representative sample of scenarios in E_1 permits conclusions about the comparative effectiveness of S_1 and S_2 in environment E_1. Observing performance on a sample of scenarios in a sample of environments from class E permits conclusions about the comparative effectiveness of S_1 and S_2 in environment class E.

Of course some variables are quite difficult to control, and these necessarily limit the conclusions that can be drawn from experiments. In particular, it is difficult to separate and control variables that distinguish among the architecture, implementation, and knowledge of a given system S. Nonetheless, if we wish to draw strong conclusions about the utility of an architecture, we must control these potentially confounding variables. In many cases, our experiments may permit conclusions only about the level of performance of an S or the relative performances of alternative Ss. It may require analytical methods–or be infeasible–to determine whether an S's performance advantage is due to its architecture, implementation, or knowledge.

other desirable E-Requirements as well). Notice, however, that other testable claims are possible, for example that a non-reactive S (one whose architecture is something different from the above-mentioned sense-act loop) also produces E-Reactivity (and perhaps some other desirable E-Requirements as well). The purpose of experiments is to evaluate such claims.

Accordingly, the model of embedded systems suggests that efforts to specify requirements for IRTPSs clearly distinguish between S-Requirements and E-Requirements. In particular, E-Requirements should be defined strictly in terms of measurements on E variables, while S-Requirements should be defined in terms of measurements on S variables. For example, S-Reactivity might be defined as a bound on the computation performed between sensing and acting. E-Reactivity might be defined as a bound on the latency between occurrence of an important "problem" event and the occurrence of an appropriate external "correction" event. Moreover, a given experiment requires agreement among participants on the E-Requirements against which alternative S-Requirements will be evaluated.

5.2 Implications for an Experimental Testbed

As discussed throughout this document, IRTPSs are complex artifacts embedded in complex environments. Experiments that allow generalization of conclusions beyond the immediate experimental conditions require control of many variables in both the S and the E. In addition, experimentation on S's of differing scopes requires flexibility in the placement of the $S - E$ boundary. To support these kinds of experimentation requires a sophisticated testbed.

For example, a basic testbed would allow one with an S to run and experiment with it. The testbed provides an E (likely simulated, and also perhaps paremeterized) and defines the $S - E$ boundary by the interface functions and data structures through which S and E interact. The testbed should also provide means for controlling multiple runs and for collecting measurements on E and, perhaps, on S as well. Ideally a testbed also provides utilities for analyzing the results (i.e., the measurements) across multiple runs.

A more general testbed facility would not have the E built in, but would instead accept both the E and the S as inputs. That is, it defines a generic interface between any E and any S compatible with that E. It would accomplish this by defining an interface between the testbed itself and S, and between the testbed and E. These interfaces would allow the testbed to

Draft for AFOSR Santa Cruz meeting

Agent Oriented Programming

Yoav Shoham
Computer Science Department
Stanford University

1 Introduction

This is an abbreviated version of a manuscript that describes work we are doing in the Artificial Agents group in the Robotics Lab. It touches on issues that are subject of much current research in AI, issues that include the relationship between a machine and its environment, and the notion of agenthood. Many of the ideas here intersect and interact with ideas of others. For the sake of continuity, however, I will delay placing this work in the context of other work until the end.

The term 'agents' is used a lot these days. This is true in AI, but also outside it, for example in connection with data bases and manufacturing automation. Although very popular, the term has been used in such diverse ways that it has become almost meaningless without reference to a particular notion of agenthood. Some notions are primarily intuitive, others quite formal. Some are very austere, defining an agent essentially as a Turing-like machine, and others ascribe to agents sensory-motor, epistemic and even natural language capabilities.

We propose viewing 'artificial agents' as formal versions of human agents, possessing formal versions of knowledge and beliefs, desires and goals, capabilities, and so on. The result is a computational framework which I will call *agent-oriented programming.*

The name is not accidental, as AOP can be viewed as an extension of the *object-oriented programming* (OOP) paradigm. I mean the latter in the spirit of Hewitt's original Actors formalism, rather than in the more technological sense in which it is used nowadays. Intuitively, whereas OOP proposes viewing a computational system as made up of modules that are able to communicate with one another and which have individual ways of handling in-coming messages, AOP expands the picture by allowing the modules to possess knowledge and beliefs about one another, to have goals and desires, and other similar notions. A computation consists of these agents informing, requesting, negotiating, competing and assisting one another.

This is the programming-paradigm perspective on AOP. An alternative view of AOP is as a formal language. From this perspective it may be viewed as a generalization of epistemic logics, which have been used a fair amount in AI and distributed computation in recent years. These logics describe the behavior of machines in terms of notions such as knowledge and belief. These mentalistic-sounding notions are actually given very precise computational meanings, and are used not only to prove properties of distributed systems, but to design them as well. A typical rule in such a 'knowledge-based' system is "if processor A does not know that processor B has received his message, then processor A will not send another message." AOP expands these logics by augmenting them with formal notions of goals, desires, capabilities, and possibly others. A typical rule in the resulting framework would be "if agent A knows that agent B intends to do something agent A does not want done, A will request that B change his intention." In addition, temporal information is included to anchor knowledge, desires and so on in particular points in time.

Intentional terms such as knowledge, beliefs, goals and so on are used in a curious sense in the formal AI community. On the one hand, the definitions (e.g., of knowledge) come nowhere close to capturing the full linguistic meanings. On the other hand, the intuitions about these formal notions do indeed derive from the everyday, commonsense meanings of the words. What is curious is that, despite the disparity, the everyday intuition has proven a good guide to employing the formal notions, when done in moderation.

The rest of this document is organized as follows. We first provide motivation for the Agent-Oriented Programming paradigm by looking at three futuristic applications. We then outline the research programme, including a sketch of the progress we have made to date. We conclude with a brief comparison with recent related research efforts.

2 Agent-Oriented Programming in the year 199x

We present three futuristic scenarios. The first involves only human agents, the second both humans and machines, and the third only robots.

2.1 Office automation

(omitted)

2.2 Manufacturing automation

Alfred and Brenda work at a car manufacturing plant. Alfred handles regular-order cars, and Brenda handles special-order ones. The plant has a welding robot, known affectionately as Calvin. The plant foreman is Dashiel. The following scenario develops.

At 8:00 Alfred requests that Calvin promise to weld ten bodies for him that day. Calvin agrees to do so. At 8:30 Alfred requests that Calvin accept the first body, Alfred agrees, and the first body arrives. Calvin starts welding it and promises to let Calvin know when it is ready for the next body. At 8:45 Brenda requests that Calvin work on a special-order car which is needed urgently. Calvin responds that it can't right then, but that it will when it finishes its current job, at approximately 9:00. At 9:05 Calvin completes welding Alfred's first car, ships it out, and offers Brenda to weld her car. Brenda ships it the car, and Calvin starts the welding. At 9:15 Alfred enquires why Calvin is not yet ready for his (Alfred's) next car. Calvin tells him why, and also that it (Calvin) expects to be ready by about 10:00. At 10:05 Calvin completes the welding of Brenda's car, and ships it out. Brenda requests that he reaccept it and do some painting, but Calvin refuses, explaining that it doesn't do painting. It then offers to weld another car for Alfred. Calvin then proceeds to weld Alfred's cars, until 12:15 at which time Brenda requests that Calvin commit to welding four more special-order cars that day. Calvin replies that it cannot, since there are still six more cars it promised to weld for Alfred that day. Brenda requests from Alfred that he allow Calvin to work on her cars first, finishing as many of Alfred's cars as it can afterwards. Alfred refuses. Brenda requests that Dashiel order Calvin to accept her Important Request and cancel its commitment to Alfred. Dashiel orders Calvin to weld two of Brenda's cars, and then as many of Alfred's as time allows.

2.3 Gofer robots

It is 1995, and the new Stanford Information Sciences building has been completed. In addition to its human inhabitants, the building is populated by about 100 Gofer robots. The role of the Gofers is to carry documents to and from the copying machine, fetch coffee and sodas, and generally make themselves useful.[1]

At a junction of corridors two Gofers, G-Ed and G-John, engage in the following exchange.

G-Ed: I intend to turn into the north corridor.

G-John: So do I; you may go first. Where are you headed?

G-Ed: The copying machine.

G-John: In that case, will you xerox this document for me and drop it off at John's office?

G-Ed: Ok.

G-Ed proceeds down the north corridor, and G-John scuttles back down the west corridor. Half way down the corridor it runs into G-Nils.

G-Nils: Where are you headed?

G-John: To John's office.

G-Nils: Watch out, there's a traffic jam in corridor #25; G-Terry and G-Mike collided, and it's not a pretty sight.

[1]The presence of the Gofers is an actual plan; the Gofer project in out Robotics Lab is currently experimenting with three platforms. The completion of the building by the said date, however, is pure speculation.

3 Research programme

The preceding scenarios made reference to mentalistic notions such as knowledge, belief, desires, goals and capabilities. The goal (...) of our research is to make engineering sense out of these abstract concepts. The result is to be a programming paradigm which we call Agent-Oriented Programming. This framework is to have three primary components.

- A restricted formal language of intentional constructs such as beliefs and goals, with clear syntax and semantics. This language will be used to define agents.
- A programming language in which to program these agents, with primitive commands such as **request** and **offer**. The semantics of the programming language will be derived from the semantics of agents.
- A compiler from the agent-level language to a machine-level language.

In the following subsections we expand on these components a little bit.

3.1 Language definition

We need to first define a precise language for talking directly about knowledge, beliefs, goals, and similar properties of agents. We have already begun work in this direction; here are a few examples of statements in the language of agents.

- The fact that at 9:00 Bob has a goal to purchase X-terminals at 10:00 is expressed by

$$\langle 9:00, G_{Bob}\langle 10:00, purchase(Bob, Xterminals)\rangle\rangle$$

- The fact that at 9:15 Alfred believes that at that time Calvin has the goal of welding car #14 at 10:00 is expressed by

$$\langle 9:15, B_{Alfred}\langle 9:15, G_{Calvin}\langle 10:00, weld(Calvin, car\#14)\rangle\rangle\rangle$$

- The fact that on Monday G-John knows that on Thursday he will know whether on Wednesday G-Ed copied document #114 on Wednesday is expressed by $\langle Monday, K_{G-John}\langle Thursday,$
$$(K_{G_{John}}\langle Wednesday, copy(G-Ed, document\#114)\rangle \vee K_{G_{John}}\langle Wednesday, \neg copy(G-Ed, document\#114)\rangle)\rangle\rangle$$

To define the language precisely its syntax and semantics must be fixed. Appendix A provides a few more details of the syntax, in its present stage of development. A fuller description of both syntax and semantics can be found in an article authored by Becky Thomas, Sarit Kraus and myself.

3.2 A programming language

The language discussed above will define the concept of artificial agents. The second step will be a development of a language in which to program these agents. Basic operations in this language will include:

- execute a primitive action;
- inform;
- request;
- consent;
- offer;
- promise;
- persuade;
- negotiate

Both preconditions and effects of these actions will refer to goals, beliefs and so on. For example, in its simplest version, a precondition of informing is that the speaker knows the information, and post conditions are that the hearer knows it, that the speaker knows that the hearer knows, and so on.

This illustrates also the extent to which we will deviate from common sense. The act of informing among human beings is very complex, and involves many considerations, such as the speaker believing that the hearer does not already know the information, the fallibility of human knowledge, and so on. We will incorporate into the formal language just as many of those features as are both needed and amenable to formalization.

As is well known from speech-act theory, the interpretation of communicative acts can be complex. For example, an apparent informing act may actually serve as a request ('that sandwich looks good'). We may wish to incorporate some of these properties into the programming language.

As of now we have no report on the design and implementation of this programming language.

3.3 Compiling into the language of machines

We have said that we intend to view machines as well as agents. If we wish to do that, however, we face the problem of bridging the gap between the AOP level and the agentless language of machines. Consider for example the task of coordinating a welding robot with other activities in the plant. We may find it convenient to speak as if the robot has "knowledge" of the cars waiting to be welded and a "goal" to weld one of them, but ultimately we need to connect to the sensors on board the welder and to its controllers.

For that we need to "compile" statements in the AOP to the machine language, so that, for example, the high-level command "offer to weld the next car" will be translated to the appropriate control commands. Of course, each machine will have highly idiosyncratic sensors and controls, and so the target language of our compiler will have to be at a slightly higher level than the individual machine. We model machines more abstractly as consisting of an internal state, input and output, and a transition function with an associated time delay. These machines can be aggregated, yielding quite complex behaviors. The compiler will take AOP expressions and commands, and translate them into input to those machines.

We propose to use the machine language outlined in our report with Barbara Hayes-Roth as our target language.

We have not yet tackled the compilation process head on. The closest we have gotten to doing so is to look at a particular aspect of the transition from high-level commands to low-level controls, namely its tolerance to small perturbations: the high level command can be reduced to more than one low level command. This comes up whether the high-level command includes intentional operators or not. For example, it is unreasonable to interpret the gross command 'roll down the middle of the corridor' as specifying a precise geometrical trajectory. Instead, we propose viewing it as a *protogram.* A protogram specifies the prototypical behavior, an ideal that is deviated from due to the interaction with other protograms (such as the 'avoid obstacles' protogram). Our current thinking about protograms is described in an appendix in the full manuscript.

4 Brief comparison with other work

Much work has been directed a t defining agents, machines and environments. Some of the work is similar to ours. In fact, some aspects of this research programme were inspired by this previous work. Other work is quite different, even though it uses similar terminology. Here I briefly mention the relationship I see to several projects in and around Stanford, given my limited knowledge of them.

- McCarthy's ELEPHANT language. Appears similar with respect to the vision of a programming language, its primitive commands, and the general wish to endow them with formal semantics.

- Winograd's project on coordination. I don't know enough to really say yet; my initial perception is that our intuitions overlap significantly, but that Winograd does not intend to rest his system on formal foundations.

- Nilsson's work on Action Nets. Similar in the desire to incorporate intentional notions into the machine model (actually, talks only of beliefs and goals). Significantly different machine model, and very different way of incorporating the intentional notions. Similar emphasis on the real-time nature of the machine.

- Genesereth's work on agents. Genesereth's model of machines inspired the one defined here, although some differences exist. In his work these machines are called agents, whereas we associate the term with the intentional level.

- Barbara Hayes-Roth's work on agents. Is similar in its general goal to combine high-level, cognitive-like behavior with real-world input and output. Emphasizes less the theoretical framework and the inter-agent aspects, and more the experimental methodology and structure of the individual agent.

- Rosenschein and Kaelbling's work on situated automata. Probably the strongest influence on our work. Differences: different intensional languages (they have no time and only the K operator, which makes reasoning at the intentional level less interesting), and also a slightly different machine-level language (no real-valued delays, as far as I can tell).

- Work, mostly at SRI, on belief, desire and intention (by Cohen, Levesque, Pollack, Konolige, Moore and others). Similar in motivation to ours as far as the semantics of agents goes. Cohen and Levesque's definition of goals is similar to (and preceded) ours, though we find that our explicit temporal framework is easier to use than the dynamic-logical language they adopt.

These are only a few of the related projects. There are others, both around Stanford and farther away. Rumor has it that some work is happening even in Massachusetts. A more detailed comparison will be good.

A A sketch of the agents language

Our language takes the notion of time as basic. Our most basic well-formed formulas (wffs) have the form

$$\langle t, p \rangle$$

where t is a time point and p is a proposition. This means that proposition p is true at time t.

If φ and ψ are wffs, then so are

- $\varphi \wedge \psi$, meaning *varphi* and ψ

- $\neg\varphi$, meaning not φ

- $\forall t\, \varphi$ (where t is a variable standing for a time point), meaning that φ is true at all times t.

We define $\varphi \vee \psi$ to mean $\neg(\neg\varphi \wedge \neg\psi)$; that is, φ or ψ, and $\varphi \rightarrow \psi$ to mean $\neg\varphi \vee \psi$; that is, φ implies ψ.

Now we must introduce our modal operators B, D, and G; these will represent belief, desire, and goals. If φ is a wff in our language, then so are

- $\langle t, B\varphi\rangle$ (where t is a constant or variable denoting a time point), meaning that at time t, φ is believed
- $\langle t, D\varphi\rangle$ (where t is a constant or variable denoting a time point), meaning that at time t, φ is desired
- $\langle t, G\varphi\rangle$ (where t is a constant or variable denoting a time point), meaning that at time t, φ is a goal

Now we need to describe some of the characteristics we expect these notions to have; for example, if we belive φ is true (at time t) and believe that $\varphi \rightarrow \psi$ is true (at the same time t) then we believe that ψ is true at time t as well. This keeps our set of beliefs internally consistent. Some properties we want to have:

B2 $\forall t\,(\langle t, B\varphi\rangle \wedge \langle t, B(\varphi \rightarrow \psi)\rangle \rightarrow \langle t, B\psi\rangle)$.

B3 $\forall t_1\, \forall t_2\, \neg\langle t_1, B\langle t_2,\, \mathit{false}\rangle\rangle$.

B4 $\forall t\,(\langle t, B\varphi\rangle \rightarrow \langle t, B\langle t, B\varphi\rangle\rangle)$.

B5 $\forall t\,(\langle t, \neg B\varphi\rangle \rightarrow \langle t, B\langle t, \neg B\varphi\rangle\rangle)$.

D1 $\forall t_1\, \forall t_2\, \neg\langle t_1, D\langle t_2, \mathit{false}\rangle\rangle$.

G1 $\forall t\,(\langle t, G\varphi\rangle \wedge \langle t, G(\varphi \rightarrow \psi)\rangle \rightarrow \langle t, G\psi\rangle)$.

G2 $\forall t\,(\langle t, G\varphi\rangle \wedge \langle t, G\psi\rangle \rightarrow \langle t, G(\varphi \wedge \psi)\rangle)$.

G3 $\forall t_1\, \forall t_2\, \neg\langle t_1, G\langle t_2,\, \mathit{false}\rangle\rangle$.

So our agents' beliefs are internally consistent and don't include falsity; agents are aware of their own beliefs (according to **B4** and **B5**; agents don't desire falsity or have it for a goal; an agent's set of goals is closed under implication and conjunction.

How do these notions interact? Whenever an agent has a goal, surely the agent also believes that it has that goal:

$$\forall t\,(\langle t, G\varphi\rangle \equiv \langle t, B\langle t, G\varphi\rangle\rangle)$$

. Intuitively, the agent must have consciously committed to act so as to bring about φ; otherwise φ would only be a desire. (****) Similarly, if an agent thinks that φ is impossible to achieve, then the agent won't choose φ as a goal, because that would mean wasting time trying to achieve something that's impossible:

$$\forall t\,\neg(\langle t, G\varphi\rangle \wedge \langle t, B\neg\varphi\rangle)$$

.

When does an agent form a goal? Presumably, only when achieving that goal will help the agent satisfy some desire the agent already has. So every goal either is also a desire, or will serve to help achieve that desire:

$$\forall t\,(\langle t, G\varphi\rangle \rightarrow \exists\psi\,\langle t, D\psi \wedge B(\varphi \rightarrow \psi)\rangle)$$

. This provides a necessary condition for having a goal; we might also specify a sufficient condition. For example, we might say that any time an agent has a desire and doesn't believe that desire is impossible to achieve, the agent must adopt that desire as a goal.

$$\forall t\,((\langle t, D\varphi\rangle \wedge \langle t, \neg B\neg\varphi\rangle) \rightarrow (\langle t, G\varphi\rangle \vee \langle t, G\neg\varphi\rangle))$$

.

Research on Intelligent Agents

Barbara Hayes-Roth
Stanford University

Paper Prepared for AFOSR Workshop on
Intelligent Real-Time Problem-Solving Systems

Santa Cruz, Ca.

October, 1989

1. Real-Time Performance in Intelligent Agents

Imagine an "errand robot" driving an automobile on its way to some destination. Noticing a yellow traffic light at the next intersection in its path, the robot infers from its current speed, distance to the light, and conservative traffic-light policy that it should stop. The robot immediately releases the accelerator and, after a few seconds, applies the brake to bring its vehicle to a gradual stop just before entering the intersection. The robot's behavior is satisfactory not simply because it produces the correct result, but because it does so at the right time. If, for example, the robot stopped very much before or after reaching the intersection, its behavior would be unsatisfactory and potentially catastrophic.

The errand robot illustrates a class of computer systems, which we call "intelligent agents," whose tasks require both knowledge-based reasoning and interaction with dynamic entities in the environment--such as human beings, physical processes, other computer systems, or complex configurations of such entities. Tasks requiring an intelligent agent occur in diverse domains, such as: power plant monitoring (Touchton88), sonar signal interpretation (Nii82), process control (Allard87, d'Ambrosio87, Fehling86, LeClair87, Moore84, Pardee87,89), experiment monitoring (O'Neill89), student tutoring (Murray89), aircraft pilot advising, and intensive care patient monitoring (Fagan80, Hayes-Roth89a).

To perform such tasks, an agent must possess capabilities for: *perception*--acquiring and interpreting sensed data to obtain knowledge of external entities; *cognition*--knowledge-based reasoning to assess situations, solve problems, and determine actions; and *action*--actuating effectors to execute intended actions and influence external entities. In the example above, the errand robot perceives signals from which it infers that the traffic light is yellow. It reasons with this perception, its traffic light policies, and other perceptions and knowledge to determine that gradually coming to a stop at the intersection is the appropriate result and that releasing the accelerator and applying the brake are the appropriate actions. It performs those actions in the appropriate temporal organization, thereby achieving the intended result.

Because external entities have their own temporal dynamics, interacting with them imposes aperiodic hard and soft real-time constraints on the agent's behavior. Following Baker89, we use the term "aperiodic" to describe tasks having irregular arrival times. Following Faulk88 and Stankovic88b, we use the terms "hard" and "soft" to distinguish between real-time constraints whose violation precludes a successful result versus those whose violation merely degrades the utility of the result. For example, a vehicle that happens to stop in front of the errand robot is an aperiodic event with a hard deadline. The robot must stop its own vehicle in time to avoid colliding with the other vehicle. When that is not possible, the robot should consider alternative actions, such as maneuvering around the stopped vehicle.

In a complex environment, an agent's opportunities for perception, action, and cognition typically exceed its computational resources. For example, in the scenario above, the errand robot has opportunities to perceive the physical features and occupants of other automobiles on the road and the buildings and landscape along the sides of the road. It might reason about any of these perceptions or other facts in its knowledge base. It might perform a variety of actions more or less related to driving its automobile. Fortunately, the robot largely ignores most of these opportunities to focus on matters related to the traffic light. Otherwise, it might fail to perform the necessary perception, reasoning, and actions in time to stop its automobile at the right time. On the other hand, the errand robot cannot totally ignore incidental information without risking the consequences of rare catastrophic events. For example, the robot should notice a child running into its path. In some cases, the robot might benefit from noticing information that is not immediately useful. For example, it might notice a sign posting business hours on a shop window and use that information when planning a subsequent day's errands.

Because an intelligent agent is almost always in a state of perceptual, cognitive, and action overload, it generally cannot perform all potential operations in a timely fashion. While faster hardware or software optimization may solve this problem for selected application systems, they will not solve the general problem of limited resources or obviate its concomitant resource-allocation task (Stankovic88a). For an agent of any speed, we can define tasks whose computational requirements exceed its resources. Moreover, we seek more from an intelligent agent than satisfactory performance of a predetermined task for which it has been optimized. Rather, we seek adaptivity of the agent to produce satisfactory performance of a range of tasks varying in required functionality and available knowledge as well as real-time constraints. For example, the errand robot should be able to respond appropriately to traffic signals and other usual and unusual events in a broad range of driving situations. It should drive competently on freeways as well as on surface streets. If it unexpectedly finds itself on surface streets where others are driving at freeway speeds (or, more likely, vice versa), it should adapt its own behavior accordingly. The agent might have other sorts of skills, such as planning its own errands under high-level goals and constraints or learning new routes from experience taking necessary detours. Other things being equal, the broader the range of tasks an agent can handle and the wider the range of circumstances to which it can adapt, the more intelligent it is.

For these reasons, we view real-time performance as a problem in intelligent control. An agent must use knowledge of its goals, constraints, resources, and environment to determine which of its many potential operations to perform at each point in time. For example, the errand robot might decide to give high priority to perceiving and reasoning about traffic lights so that it can always stop in time for yellow or red lights. When the operations required to achieve an agent's current goals under its specified constraints exceed its computational resources, it may have to modify them as well. For example, if the errand robot finds itself unexpectedly late to an important destination, it might decide to relax its conservative traffic-light policy and drive through selected yellow lights. Because it is situated in a dynamic environment and faces a continuing stream of events, an agent must make a continuing series of control decisions so as to meet demands and exploit opportunities for action as they occur. For example, if the errand robot is making a planned gradual stop at a traffic light and a child runs into its path, the robot should perceive the child and stop immediately. In general, an agent should use intelligent control to produce the best results it can under real-time constraints and other resource (e.g., information, knowledge) constraints.

Our conception of real-time performance in intelligent agents is qualitatively different from conceptions of real-time performance in other sorts of computer systems (Baker89, Brinkley89, Faulk88, Henn89, Lauber89, Marsh86). In particular, we do not view real-time performance as a guaranteed, universal, or provable property of the agent. Nor do we seek real-time performance through effective engineering of the agent. We feel that these constructs are surely premature and possibly unrealistic for the versatile and highly adaptive agents we envision. Rather, we view real-time performance as one of an agent's several objectives, which it will achieve to a greater or lesser degree as the result of interactions between the environment it encounters, the resources available to it, and the decisions it makes. In many cases, the agent will achieve real-time performance only at the expense of quality of result or by compromising response quality or real-time constraints on other tasks. Ironically, as the agent's competence expands, so will its need to make such compromises. From this perspective, real-time performance in intelligent agents depends critically upon an underlying architecture that enables agents to make and apply the necessary kinds of control decisions.

2. Real-Time Requirements and Heuristics

An intelligent agent's real-time requirements can be summarized very simply: To maximize the number of important goals for which it achieves an acceptable result at an acceptable time. This section presents heuristics for meeting these requirements. We do not mean to suggest that these heuristics provide the optimal approach to meeting the requirements or even a valid approach. We suggest only that they represent a promising approach, which we currently are investigating.

1. *Asynchrony.* For a given objective, an agent can't count on all necessary perceptual information being available at the start of its associated reasoning or on its reasoning being completed prior to execution of its first associated action. Relevant information may arrive at any time during reasoning and relevant reasoning may continue beyond initiation of early actions. In addition, with multiple objectives, the agent may have to interleave unrelated perception, reasoning, and action operations. Finally, the agent must always be prepared to interrupt its ongoing acitivites to handle unpredictable emergencies or simply to switch its attention to more important matters than those currently under consideration. For these reasons, an intelligent agent must perceive, reason, and act asynchronously, without these activities interrupting or directly interfering with one another or with communications among them. For example, in the scenario above, the errand robot's execution of actions planned to produce a gradual stop do not impede its immediate perception of a child running unexpectedly into its path or its consequent actions to avoid hitting the child.

2. *Timeliness.* Because an agent interacts with independent dynamic entities, its sensory information is perishable, the utility of its reasoning operations degrades with time, and the efficacy of its actions depends upon synchronization with fleeting external events. Therefore, the agent must perceive present or recent sensed events, perform present or recent reasoning operations, and execute currently intended actions, regardless of how many earlier unexploited opportunities have occurred. The agent should not fall behind real time to handle a backlog of inputs in any of these categories and it should not operate on seriously out of date inputs--unless it has explicitly decided to do so. For example, while approaching an intersection, the errand robot perceives information relevant to its reasoning about actions to be taken at the intersection. Having passed

through the intersection, the robot ordinarily does not dwell upon unprocessed perceptual events available at the intersection, inferences it might have drawn, or alternative actions it might have taken. Instead, it performs operations related to its current post-intersection situation.

3. *Selectivity.* Except for rare occasions, opportunities for perception, cognition, and action vastly exceed the agent's resources for performing those operations. Even when the rate of such opportunities is well within the agent's capacity, it may be unproductive or even harmful for the agent to pursue many of them. Therefore, the agent must selectively perceive information and perform reasoning operations that enable it to perform the most useful actions. For example, although the errand robot could respond to great quantities of incidental information regarding the other vehicles on the road, their drivers' intentions, or the passing landscape, it ignores most of these opportunities in favor of activities related to its current driving task.

4. *Coherence.* Most non-trivial tasks require coordinated perception, cognition, and action. In the simplest case, perceived information must be integrated with knowledge and reasoning to determine actions that lead to objectives. In more complex, but no less typical cases, achievement of objectives depends upon a strategic sequence of such activities coordinated over a period of time. To perform such tasks, the agent must develop strategic plans and use them in a "top-down" fashion to direct its perception, cognition, and action toward the desired objectives. For example, given its general plan for travelling about the city and dispatching errands, the errand robot focuses on dynamically refining its intended route and obeying traffic laws as it follows that route.

5. *Flexibility.* A dynamic environment entails considerable uncertainty in the situations an agent will face and the details of those situations as they unfold. In addition to rare catastrophic events, many unanticipated events simply require minor modifications of the agent's behavior or even offer new opportunities for the agent to improve the overall utility of its performance. In order to adapt to actual evolving situations, the agent must monitor the environment (and its own inferences as well) for important unanticipated events and respond flexibly to those events in a "bottom-up" fashion to modify its objectives or its strategic plans for achieving them. For example, to avoid traffic delays caused by an accident, the errand robot should modify its planned route and take a reasonable detour. If the robot calculates that the detour also entails excessive delay, it might eliminate certain errands from its plan in order to insure completion of critical errands in the time available.

6. *Responsivity.* Situations vary in urgency--that is, the rapidity with which a response is required in order to produce a satisfactory result. Some situations impose "hard" deadlines; a logically correct response that occurs after the deadline is as useless as a logically incorrect response. Other situations impose "soft" deadlines; the utility of a logically correct response declines monotonically, but not precipitously, with delay after deadline. Depending upon the complexity of its environment, the agent may incur multiple aperiodic tasks imposing various hard and soft deadlines within a local time interval. Often it must compromise the correctness or timeliness of its performance of some of these tasks in order to satisfy the requirements of other more important tasks. However in general, other things being equal, the more urgent a situation is, the more quickly the agent should perceive the relevant information, perform the necessary reasoning, and execute the appropriate actions. Thus, the errand robot should perceive a child in its path and stop its vehicle as quickly as possible, while a somewhat longer latency is tolerable for perceiving and stopping at a yellow traffic light.

7. *Robustness*. Despite its best efforts to use its limited resources wisely, an agent will encounter situations that strain or exceed its capacity: too many important perceptions, reasoning tasks, and actions. Regardless of the degree of overload, the agent must *continue to function*. It cannot arbitrarily ignore pending tasks or fail to complete tasks it has decided to undertake. Instead, the agent must adapt to high resource-stressing situations by ensuring graceful degradation of its performance. It must reduce the demands on its resources by eliminating or revising the least important of its objectives and refocusing its resources where they are needed most. It must compromise the quality of its performance to provide satisfactory results for the most important objectives. For example, when driving near a school or playgound, the errand robot may compromise its responsivity to traffic lights so that it can monitor more vigilantly for children running into its path.

3. Research Goals

The primary goal of our research is to develop a general architecture for *intelligent agents*--systems that perform a variety of knowledge-based reasoning tasks while functioning autonomously in natural environments. By definition, an intelligent agent must integrate capabilities for performing the following tasks: interpretation of information sensed from a complex, dynamic environment; detection and diagnosis of exceptional conditions; reactive response to urgent conditions; prediction of important future conditions; planning and execution of a course of actions to influence external conditions; explanation of the physical phenomena underlying observed or predicted or planned conditions; consultation with human beings or other computer systems; and learning to improve performance based on experience. In performing these several tasks, the agent must allocate limited computational and other resources (e.g., sensors and effectors) dynamically, to achieve its most important objectives in a timely fashion. In particular, they must achieve at least the real-time performance objectives characterized above.

4. Architecture

In previous work, we developed a blackboard architecture that integrates task-level reasoning with reflective processes, such as dynamic planning and control of reasoning, strategic explanation, and strategy learning. Implemented in the *BB1* system, this architecture has been used in a number of projects at Stanford and licensed widely outside of Stanford. Our current work is directed toward extending the architecture in four areas: perception/action processes to mediate asynchronous interactions with a complex environment, a satisficing control regime to support guaranteed response times, and conceptual graph representation to provide a declarative representation for all of an IRTPS's knowledge, beliefs, intentions, etc., with a temporally organized representation of dynamic information. The architecture is designed to requirements for dynamic real-time control. Current research activities are intended to evaluate its achievement of those design goals both analytically and empirically. The architectural design and implementation and related research are discussed in several articles in the attached references, especially Hayes-Roth85, 87b, 88a, 89c.

We are developing generic models of knowledge and reasoning for prototypical tasks performed by an intelligent agent. Examples are: focus of attention, incremental modeling of dynamic external phenomena; reactive detection, diagnosis, and correction of exceptional external conditions; model-based diagnosis, prediction, and explanation of external conditions; time-constrained planning of longer-term courses of action. Our approach represents explicit knowledge of the operations and strategies involved in each of these tasks within the architecture discussed above. Some of this work is discussed in (Boureau89, Hewett89b, Hayes-Roth89b, Washington89).

Much of our past work concerns coordination of diverse knowledge sources and reasoning methods invoked by run-time conditions (Garvey87, Hayes-Roth85, Johnson87) to achieve efficient and effective performance of complex tasks. Current research expands these concerns to performance of multiple interacting tasks under real-time and other resource constraints. In particular, we have developed an approach to integrating reactive response to urgent events in the context of more deliberate situation assessment and planning for stable or slowly evolving conditions.

5. Experimental Applications

Because our long-term research goal is to develop a general architecture for adaptive intelligent systems, much of our research involves experimental development of application systems in various several domains. Each new domain tests the efficacy and generality of the current architecture and presents new requirements to be addressed in subsequent versions of the architecture. We currently are working on several intelligent real-time monitoring applications, described briefly below.

Guardian, being developed in collaboration with Dr. Adam Seiver, of the Palo Alto Veterans Administration Medical Center, monitors ventilator-supported patients and consults with physicians and nurses in a surgical intensive care unit. Demonstration 4, which will be completed this Fall, monitors about twenty automatically sensed variables (e.g., pressures, temperature) and a few irregularly sensed variables (e.g., lab results). It performs the several tasks mentioned above using several kinds of knowledge: heuristic knowledge related to common respiratory problems; structure/function knowledge of the respiratory, circulatory, metabolic, and mechanical ventilator systems; and structure/function knowledge of generic flow, diffusion, and metabolic systems. Guardian currently monitors a simulated patient-ventilator-hospital system. However, beginning this winter, we expect to have on-line access to patient data monitors at the Palo Alto VAMC, so that we can begin experiments involving real-time monitoring of actual patients. A paper describing Guardian is in preparation.

A second system, being developed in collaboration with Professor James Harris of Stanford's Electrical Engineering Department, is intended to control the growth of gallium arsenide (GaAs) crystals by molecular beam epitaxy (MBE). A simulation of the MBE machine and crystal growth process have been developed and a preliminary process planner is near completion.

We currently are exploring two other domains: intelligent plant monitoring for preventive maintenance, in collaboration with Professors Raymond Levitt and Paul Teicholz of Stanford's Civil Engineering Department; and intelligent control of micro-factory operations for semiconductor fabrication, in collaboration with Professor Nils Nilsson of Stanford's Computer Science Department and with Professors Krishna Saraswat, Gene Franklin, and Robert Dutton of Electrical Engineering.

6. Related Work

In this section, we briefly review other research related to adaptive intelligent systems. This is by no means a complete survey, but gives an indication of some of the most prominent lines of relevant research and their treatment of the requirements discussed above.

A number of researchers have extended the classical planning model (Fikes71, Sacerdoti75) to permit interleaving of planning and execution, either to build a plan incrementally or to modify the plan in response to unanticipated conditions (Broverman87, Corkill82, Durfee86, Georgeff87, Hayes-Roth85, Lesser88) or to employ more knowledge intensive and computationally tractable methods for generating partial plans, including for example: instantiating goal-oriented action schemas (Friedland79); integrating top-down and bottom-up planning methods (Hayes-Roth79a,b, Johnson87), transferring prior successful plans to new situations (Hammond86); or successively applying constraints among potential actions (Stefik81)). These approaches are quite compatible with our work.

Several researchers have studied methods for controlling trade-offs in the amount of time spent solving a problem or performing a task versus the quality of the result, (Dean88, Horvitz87, Lesser88). These approaches, which Dean calls "anytime algorithms" and Lesser calls "approximate reasoning," are quite compatible with and appear in several aspects of the proposed architecture, notably in its satisficing control regime and in its accommodation of alternative reasoning strategies, including approximate or anytime strategies.

By contrast, in an effort to avoid the computational cost of control reasoning and thereby create real-time responsivity, a number of researchers have turned their attention to the theory, design, and implementation of "reactive agents" (Agre87, Andersson88, Brooks85, Fagan84, Firby87, Kaelbling88, Nilsson89, Rosenschein86, Schoppers87, Suchman87). Basically, reactive agents store large numbers of perception-action rules in a computationally efficient form and execute actions invoked by environmental conditions on each iteration of a perceive-act cycle. Thus, they are formally similar to control theoretic methods (Bollinger88, Hale73), where traversal of symbolic networks replaces computation of numerical models. We consider reactivity an essential behavior in some circumstances and our architecture supports it, but we do not consider it a promising foundational architectural principle, first because enumerating all possible perception-action contingencies and encoding them in a computationally tractable form is infeasible for challenging task domains and second because many task domains intrinsically require maintenance of internal state.

For several decades now, robotics researchers have aimed to build "task-level" robot systems (Cox89, Ernst61, Kathib86, Lozano-Perez89). Unlike robots programmed to perform specific mechanical tasks, task-level robots are intended to accept high-level goals and then determine and perform whatever behaviors are necessary to achieve the goals. They are intended to operate under a variety of incidental contextual conditions, including low-frequency exceptional conditions related to hardward, software, or environmental state. Significant applications of this work include efforts to build autonomous vehicles (Burks89, Crowley89, Goto89, McTamaney89). The robotics work is similar in spirit to the present approach--integrating perception, action, and cognition to achieve goals in a real-time task environment--differing primarily in emphasis on perceptual-motor rather than cognitive functions. The robotics research traditionally has focused on challenging perceptual-motor tasks, but has begun to incorporate more cognitive activities, such as goal determination, planning, exception handling, and learning (Bares89, Barhen89, Rao89, Weisbin89). Conversely, our work grows out of earlier work emphasizing reasoning and problem solving, with new emphases on perceiving and acting in a real-time environment.

Finally, there has been interesting theoretical work aimed at a formal characterization of agents (e.g., Genesereth89). This work tends to encompass an extremely broad spectrum of computational entities. However, recent work by Shoham (described in his paper for this workshop) focuses on agents possessing formal versions of knowledge, beliefs, desires, goals, etc., bringing it closer in spirit to the kinds of intelligent agents discussed in this paper.

Hayes-Roth, B. Architectural foundations for real-time performance in intelligent agents. *Journal of Real-Time Systems*, accepted for 1990 publication pending revision.

Hayes-Roth, B., Washington, R., Hewett, R., Hewett, M., and Seiver, A. Intelligent monitoring and control. *Proceedings of the International Joint Conference on Artificial Intelligence*, Detroit,1989.

Washington, R., and Hayes-Roth, B. Input Data Management in Real-Time AI Systems. *Proceedings of the International Joint Conference on Artificial Intelligence*, Detroit,1989.

Hayes-Roth, B. Making intelligent systems adaptive. In K. VanLehn (Ed.), Architectures for Intelligence. Lawrence Erlbaum, 1989.

Hayes-Roth, B. Dynamic control planning in adaptive intelligent systems. *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, 1987

Hayes-Roth, B. A blackboard architecture for control. *Artificial Intelligence* , 1985, 26, 251-321. Reprinted in: Bond, A., and Gasser, L. (Eds.), Readings in Distributed Artificial Intelligence, Morgan Kaufmann Publishers, Inc., 1988.